Struktury danych i złożoność obliczeniowa
Projekt

15/06/2021

252713 Tomasz Rzymyszkiewicz

(3) Algorytmy grafowe

Spis treści

1 Sformułowanie zadania

Zadanie ma na celu implementację struktur reprezentujących grafy oraz algorytmów wyznaczających minimalne drzewo rozpinające oraz wyszukujących najkrótsze ścieżki w grafie. Po uzgodnieniu z prowadzącym zadanie zostało ukierunkowane na zbadanie obu algorytmów tylko na dwóch strukturach - macierzy sąsiedztwa oraz liście sąsiedztwa.

W programie została również zaimplementowana struktura macierzy incydencji jednak nie zostały wykonane algorytmy wykorzystujące tą strukturę. W przypadku badanych algorytmów Jarnika-Prima oraz Dijkstry wydaje się ona mieć mniej praktyczne zastosowanie w stosunku do macierzy sąsiedztwa lub listy sąsiedztwa.

1.1 Algorytm Jarnika-Prima

Algorytm Jarnika-Prima ma na celu wyznaczenie minimalnego drzewa rozpinającego (ang. *MST*) tj. grafu acyklicznego spójnego zawierającego wszystkie wierzchołki grafu, którego krawędzie mają najmniejszą możliwą wagę. Jest to algorytm zachłanny. Teoretyczna złożoność czasowa tego algorytmu wynosi $O(|E| * |V|)$, gdzie $|E|$ to liczba krawędzi grafu, a $|V|$ to liczba wierzchołków grafu. Wynika ona z konieczności rozpatrzenia $|V| - 1$ wierzchołków oraz sprawdzenia krawędzi wchodzących z wierzchołków dołączonych do drzewa, których w najgorszym przypadku może być $|E|$.

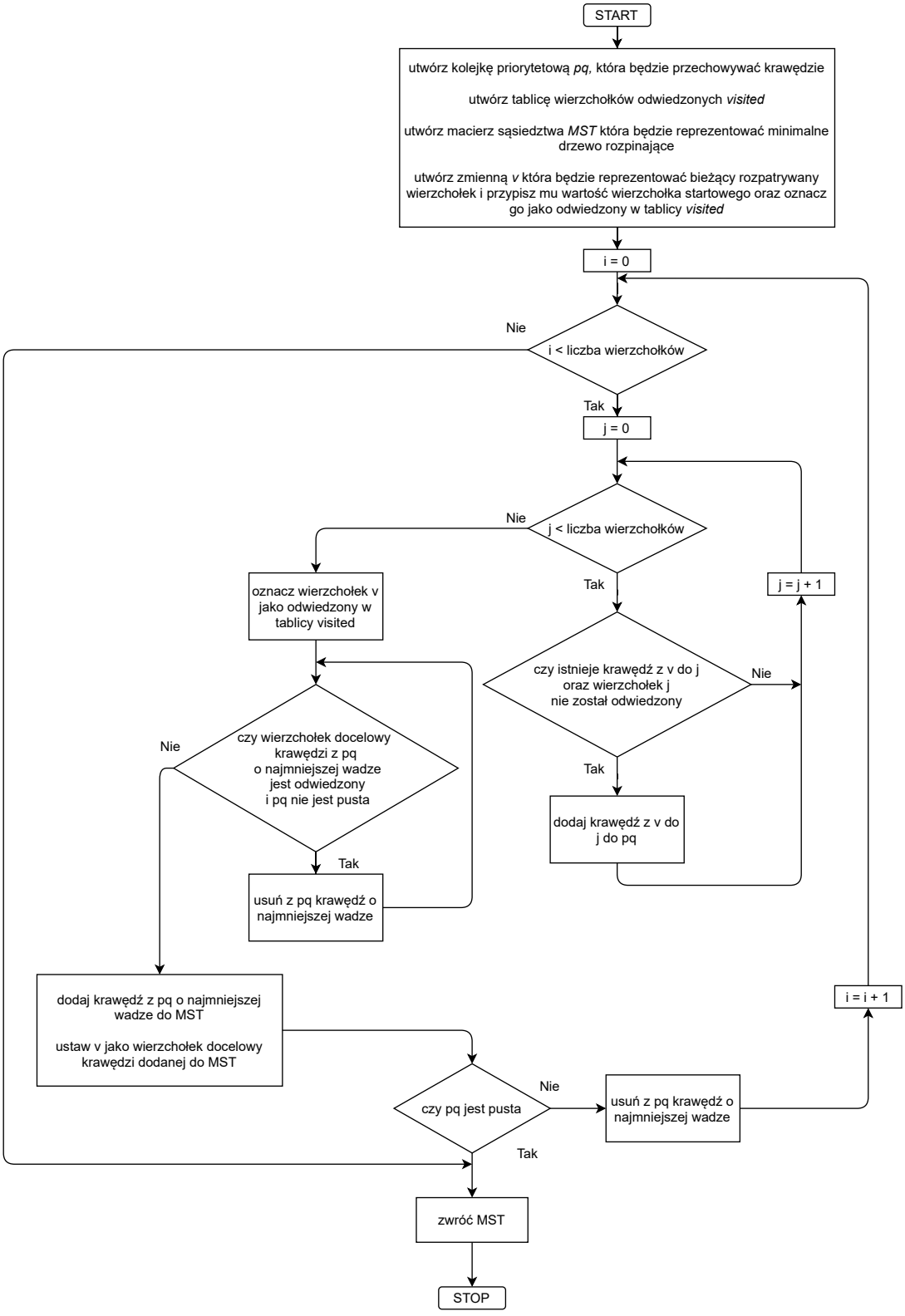
1.2 Algorytm Dijkstry

Algorytm Dijkstry ma na celu wyznaczyć najkrótsze (takie których łączna suma wag jest najmniejsza) ścieżki (uporządkowane zbiory krawędzi) z wybranego wierzchołka początkowego do wszystkich pozostałych wierzchołków w grafie. Jest to algorytm zachłanny. Algorytm daje prawidłowy wynik tylko dla grafu o nieujemnych wartościach wag. Teoretyczna złożoność czasowa tego algorytmu wynosi $O(|V|^2 * |E|)$. Wynika to z konieczności wykonania $|V|$ iteracji w których wybieramy nieodwiedzony wierzchołek o najmniejszej wadze dojścia przeglądając tablicę wierzchołków a następnie rozpatrujemy krawędzie z niego wychodzące, których w najgorszym przypadki może być $|E|$. Oparcie algorytmu na kopcu mogłoby zoptymalizować algorytm do złożoności $O(|V| \log |V| * |E|)$, jednak nie zostało to zaimplementowane w badanym przypadku.

2 Algorytmy

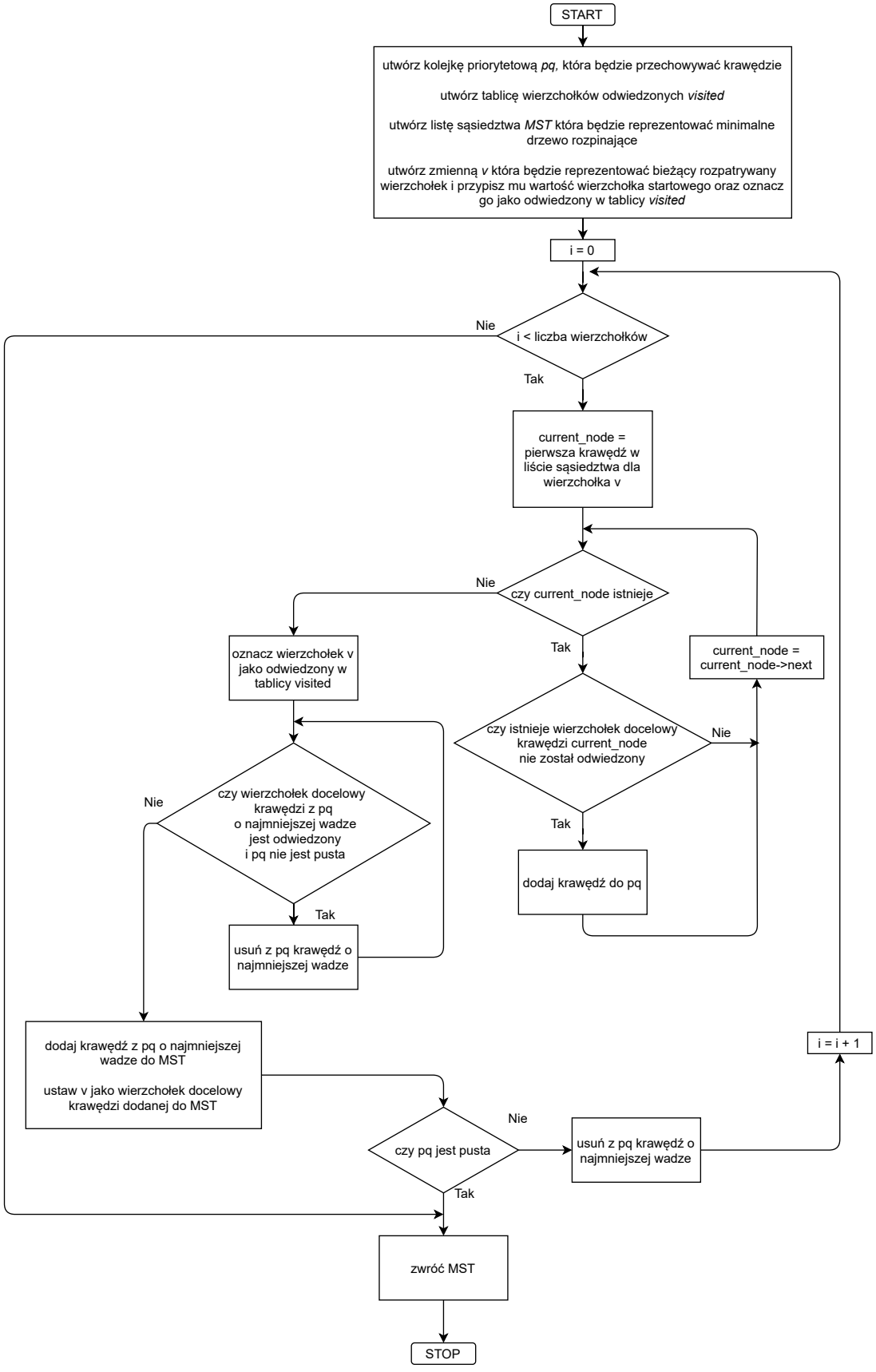
2.1 Algorytm Jarnika-Prima

2.1.1 Macierz sąsiedztwa



Rysunek 1: Schemat blokowy algorytmu wyznaczania minimalnego drzewa rozpinającego (z wykorzystaniem algorytmu Jarnika-Prima) grafu w reprezentacji macierzy sąsiedztwa.

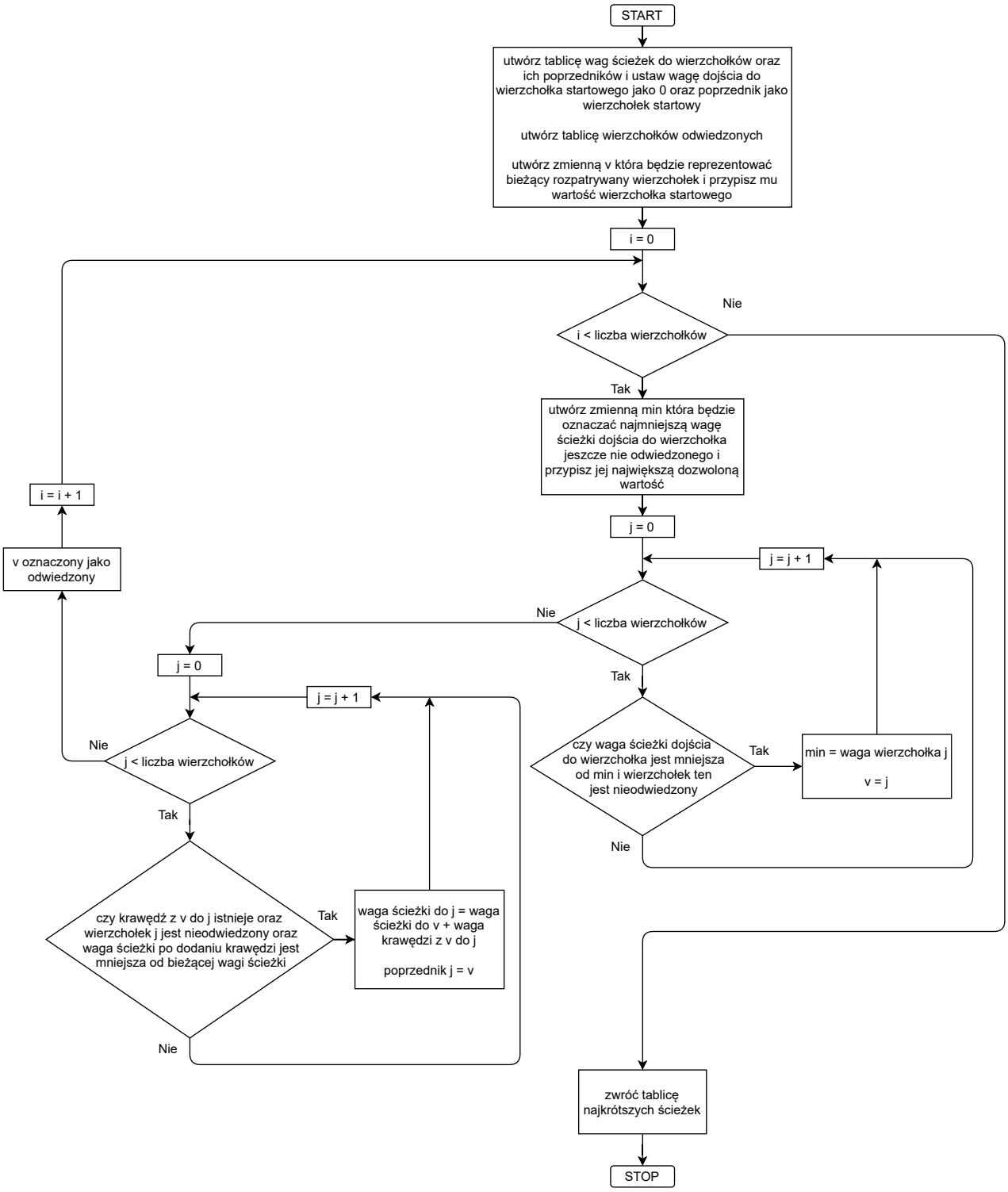
2.1.2 Lista sąsiedztwa



Rysunek 2: Schemat blokowy algorytmu wyznaczania minimalnego drzewa rozpinającego (z wykorzystaniem algorytmu Jarnika-Prima) grafu w reprezentacji listy sąsiedztwa.

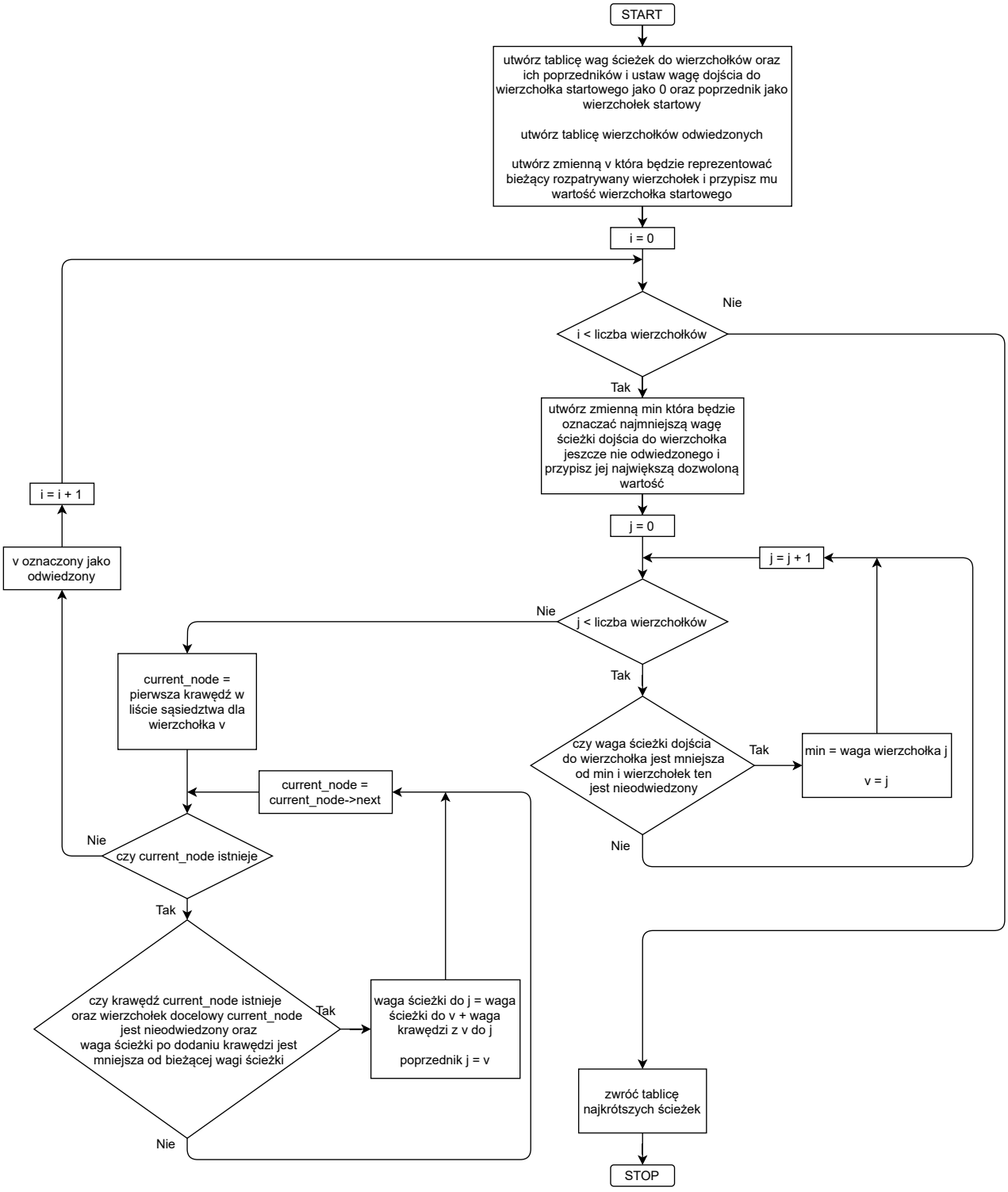
2.2 Algorytm Dijkstry

2.2.1 Macierz sąsiedztwa



Rysunek 3: Schemat blokowy algorytmu wyznaczania najkrótszych ścieżek (z wykorzystaniem algorytmu Dijkstry) grafu w reprezentacji macierzy sąsiedztwa.

2.2.2 Lista sąsiedztwa



Rysunek 4: Schemat blokowy algorytmu wyznaczania najkrótszych ścieżek (z wykorzystaniem algorytmu Dijkstry) grafu w reprezentacji listy sąsiedztwa.

3 Dane wejściowe

3.1 Generowanie grafów

Generowanie grafów wykonywane jest za pomocą skryptu napisanego w języku Python, wykorzystującego bibliotekę metodę `fast_gnp_random_graph` z biblioteki `networkx`. Metoda ta tworzy graf Erdősa–Rényiego który jest grafem $G(n, p)$ gdzie n to liczba wierzchołków, a p to prawdopodobieństwo dołączenia krawędzi do grafu (prawdopodobieństwo jest niezależne dla każdej krawędzi).

Po wygenerowaniu grafu do każdej krawędzi przypisywana jest waga o losowo wybranej wartości w zadanym zakresie.

3.1.1 Plik konfiguracyjny `adv-data-config.ini`

Jest to plik konfiguracyjny wczytywany przez program `generate_graph_advanced.py` w celu wygenerowania zadanego grafu.

Składnia pliku konfiguracyjnego:

```
[graph]
graph_name = <nazwa grafu>
vertices = <liczba wierzchołków>
density = <gęstość grafu w zakresie od 0 do 1>
min_weight = <minimalna waga krawędzi>
max_weight = <maksymalna waga krawędzi>
```

3.2 Wykonywanie algorytmów oraz pomiar czasu wykonywania i zajętości pamięci

Wykonywanie algorytmów odbywa się przy pomocy programu `graphs.cpp`. Umożliwia on wczytanie grafów z pliku CSV do wybranej struktury oraz wykonanie algorytmu Jarnika-Prima lub Dijkstry. Wykonanie algorytmu może odbyć się w 3 trybach:

- wykonanie algorytmu i wypisanie wyniku na ekran
- wykonanie algorytmu, pomiar czasu wykonania i zapis wyniku do pliku CSV
- wykonanie algorytmu, pomiar zajętości pamięci i zapis wyniku do pliku CSV

3.2.1 Plik konfiguracyjny `config.ini`

Jest to plik konfiguracyjny wczytywany przez program `graphs.cpp` w celu wykonania określonych w treści pliku zadań.

Składnia pliku konfiguracyjnego:

```
<nazwa pliku wyjściowego (CSV)>
<nazwa pliku z grafem> <struktura reprezentująca graf> <rodzaj grafu> <algorytm> <liczba powtórzeń> <rodzaj operacji>
...
```

Określone pola w pliku konfiguracyjnym mogą przyjmować następujące wartości:

- `<nazwa pliku wyjściowego (CSV)>` - dowolna nazwa pliku CSV
- `<nazwa pliku z grafem>` - nazwa pliku z grafem w formacie listy krawędzi z wagami
- `<struktura reprezentująca graf>` - `adjacency_list` lub `adjacency_matrix`

- `<rodzaj grafu>` - `directed` lub `undirected`
- `<algorytm>` - `MST` lub `shortest_paths`
- `<liczba powtórzeń>` - ile razy algorytm będzie powtórzony w czasie pomiaru
- `<rodzaj operacji>` - `print` lub `time` lub `memory`

3.3 Dane wyjściowe

Wyniki przeprowadzonych pomiarów zapisywane są do zadanego pliku CSV z wyszczególnionymi nazwami kolumn:

- `structure` - struktura reprezentująca graf
- `graph_type` - czy graf był skierowany czy nieskierowany
- `operation` - rodzaj algorytmu
- `number_of_vertices` - liczba wierzchołków grafu
- `number_of_edges` - liczba krawędzi grafu
- `time_memory` - czas wykonania operacji w sekundach lub zajętość pamięci w bajtach
- `number_of_repeats` - liczba powtórzeń operacji
- `test_type` - rodzaj testu `time` lub `memory`

4 Procedura badawcza

4.1 Algorytm Jarnika-Prima

4.1.1 Macierz sąsiedztwa

Badanie wpływu liczby wierzchołków w grafie (w reprezentacji macierzy sąsiedztwa) na czas wykonywania algorytmu Jarnika-Prima została podzielona na dwa etapy. W pierwszym etapie badano grafy o gęstości 0,3 i liczbie wierzchołków w zakresie od 100 do 1000 wierzchołków. W drugim etapie były to grafy o gęstości 0,7 i liczbie wierzchołków w zakresie od 100 do 1000. Pomiary powtórzono 100 razy.

4.1.2 Lista sąsiedztwa

Badanie wpływu liczby wierzchołków w grafie (w reprezentacji listy sąsiedztwa) na czas wykonywania algorytmu Jarnika-Prima została podzielona na dwa etapy. W pierwszym etapie badano grafy o gęstości 0,3 i liczbie wierzchołków w zakresie od 100 do 1000 wierzchołków. W drugim etapie były to grafy o gęstości 0,7 i liczbie wierzchołków w zakresie od 100 do 1000. Pomiary powtórzono 100 razy.

4.2 Algorytm Dijkstry

4.2.1 Macierz sąsiedztwa

Badanie wpływu liczby wierzchołków w grafie (w reprezentacji macierzy sąsiedztwa) na czas wykonywania algorytmu Dijkstry została podzielona na dwa etapy. W pierwszym etapie badano grafy o gęstości 0,3 i liczbie wierzchołków w zakresie od 100 do 1000 wierzchołków. W drugim etapie były to grafy o gęstości 0,7 i liczbie wierzchołków w zakresie od 100 do 1000. Pomiary powtórzono 100 razy.

4.2.2 Lista sąsiedztwa

Badanie wpływu liczby wierzchołków w grafie (w reprezentacji listy sąsiedztwa) na czas wykonywania algorytmu Dijkstry została podzielona na dwa etapy. W pierwszym etapie badano grafy o gęstości 0,3 i liczbie wierzchołków w zakresie od 100 do 1000 wierzchołków. W drugim etapie były to grafy o gęstości 0,7 i liczbie wierzchołków w zakresie od 100 do 1000. Pomiary powtórzono 100 razy.

4.3 Pomiar zajętości pamięci

4.3.1 Algorytm Jarnika-Prima

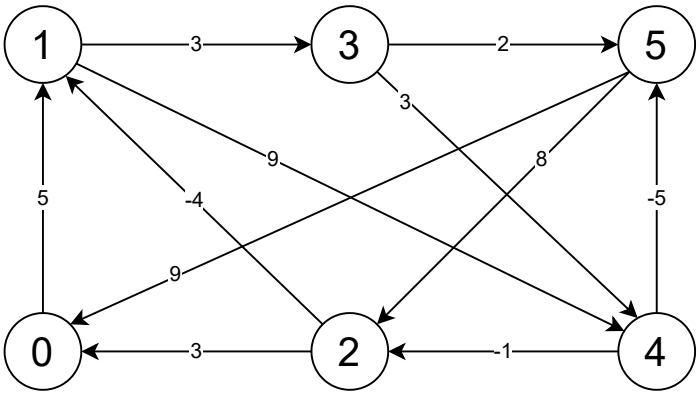
Badanie wpływu liczby wierzchołków w grafie (w reprezentacji listy sąsiedztwa oraz macierzy sąsiedztwa) na zajętość pamięci w czasie wykonywania algorytmu Jarnika-Prima zostało wykonane na grafie o gęstości 0,3 i liczbie wierzchołków w zakresie od 100 do 1000 wierzchołków.

4.3.2 Algorytm Dijkstry

Badanie wpływu liczby wierzchołków w grafie (w reprezentacji listy sąsiedztwa oraz macierzy sąsiedztwa) na zajętość pamięci w czasie wykonywania algorytmu Dijkstry zostało wykonane na grafie o gęstości 0,3 i liczbie wierzchołków w zakresie od 100 do 1000 wierzchołków.

4.4 Badanie wpływu ujemnych wag na poprawność wykonywania algorytmu Dijkstry

W celu sprawdzenia poprawności działania algorytmu Dijkstry na grafie z wagami ujemnymi przeprowadzono próbę wyznaczenia najkrótszych ścieżek w poniższym grafie.

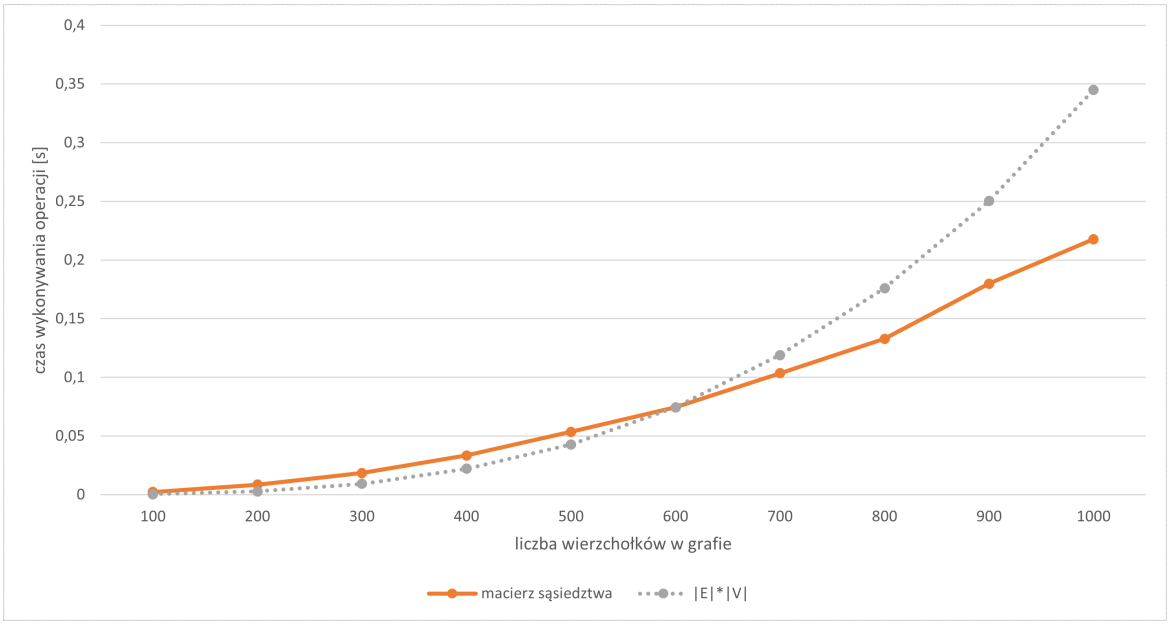


Rysunek 5: Przykładowy graf z wagami ujemnymi (bez cykli ujemnych).

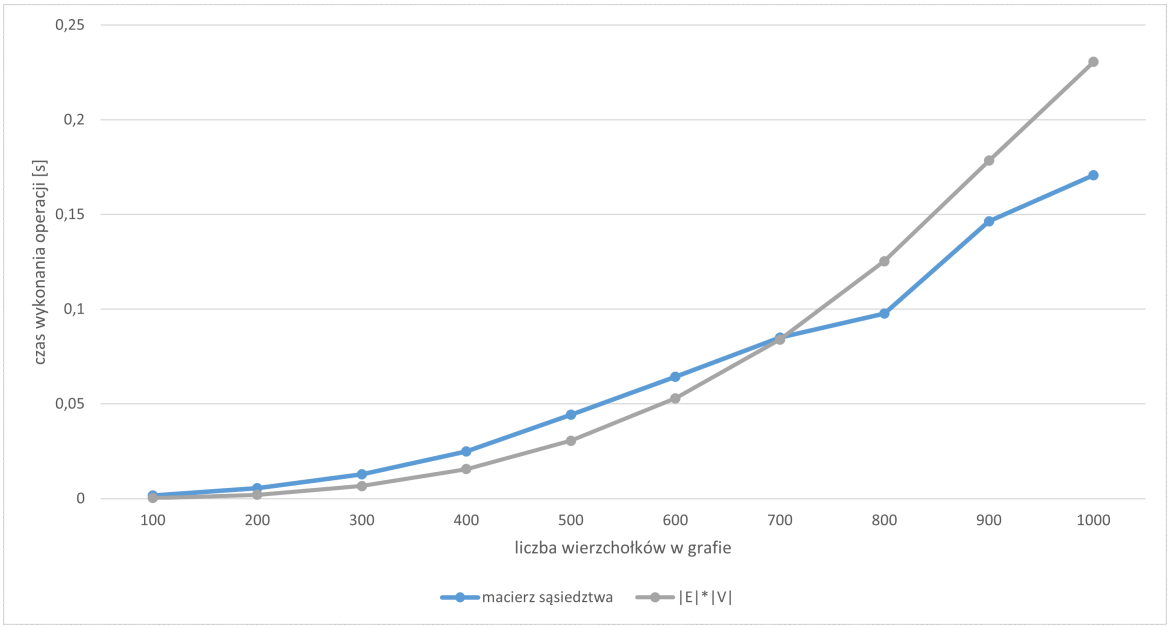
5 Wyniki pomiarów czasu wykonywania algorytmów

5.1 Algorytm Jarnika-Prima

5.1.1 Macierz sąsiedztwa

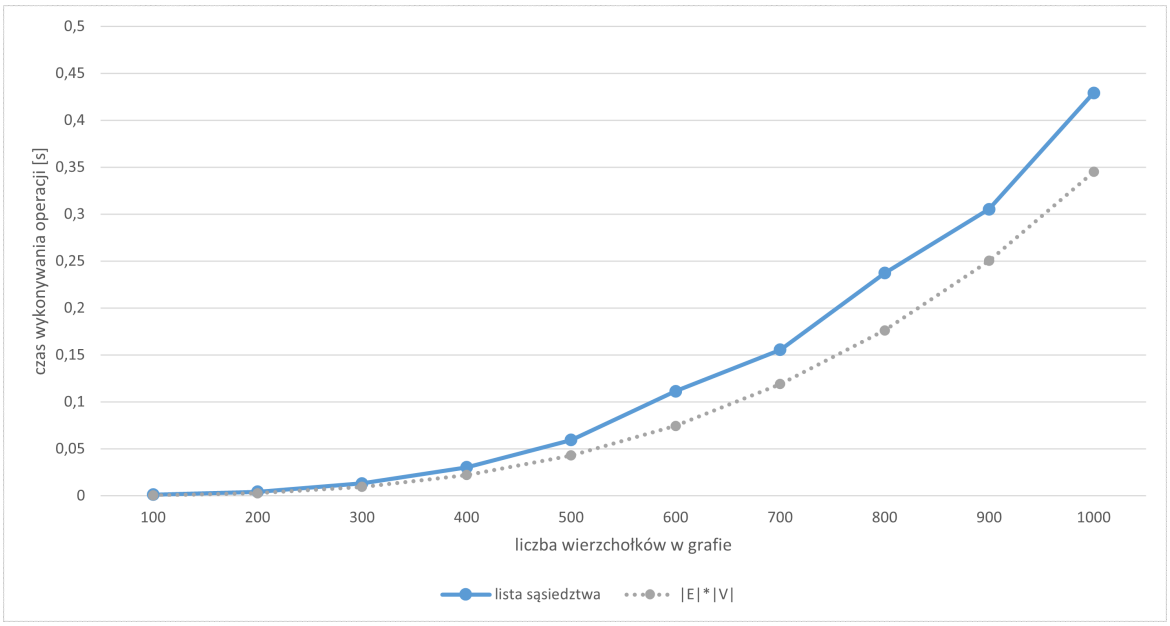


Rysunek 6: Wpływ liczby wierzchołków w grafie o gęstości 0,3 na czas wykonywania algorytmu Jarnika-Prima w grafie na reprezentacji macierzy sąsiedztwa.

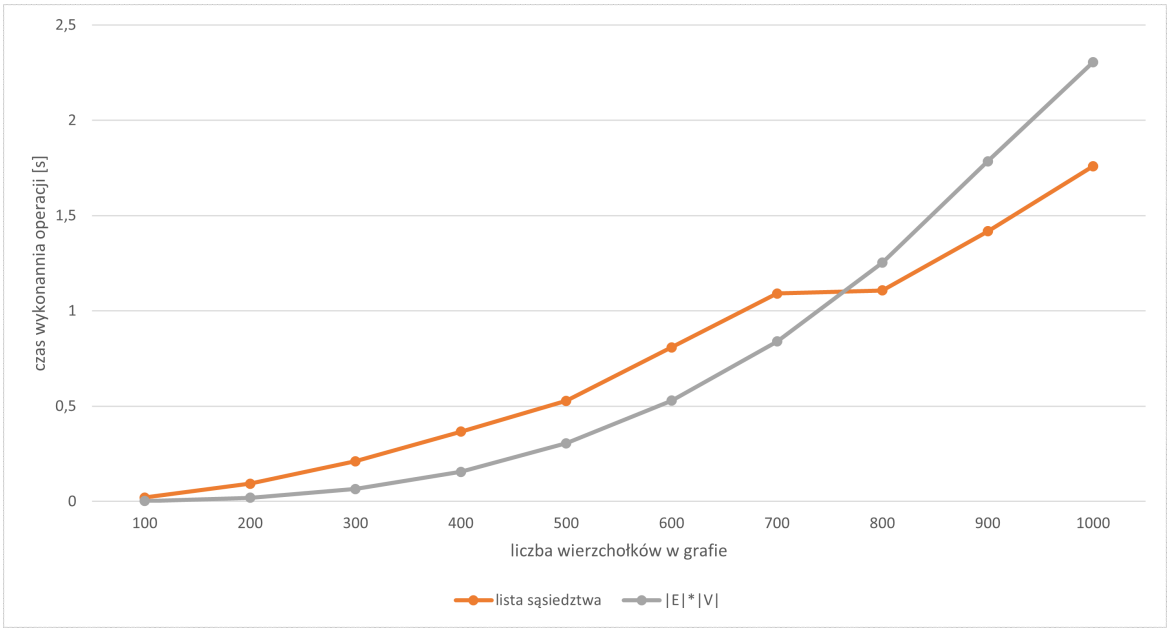


Rysunek 7: Wpływ liczby wierzchołków w grafie o gęstości 0,7 na czas wykonywania algorytmu Jarnika-Prima w grafie na reprezentacji macierzy sąsiedztwa.

5.1.2 Lista sąsiedztwa



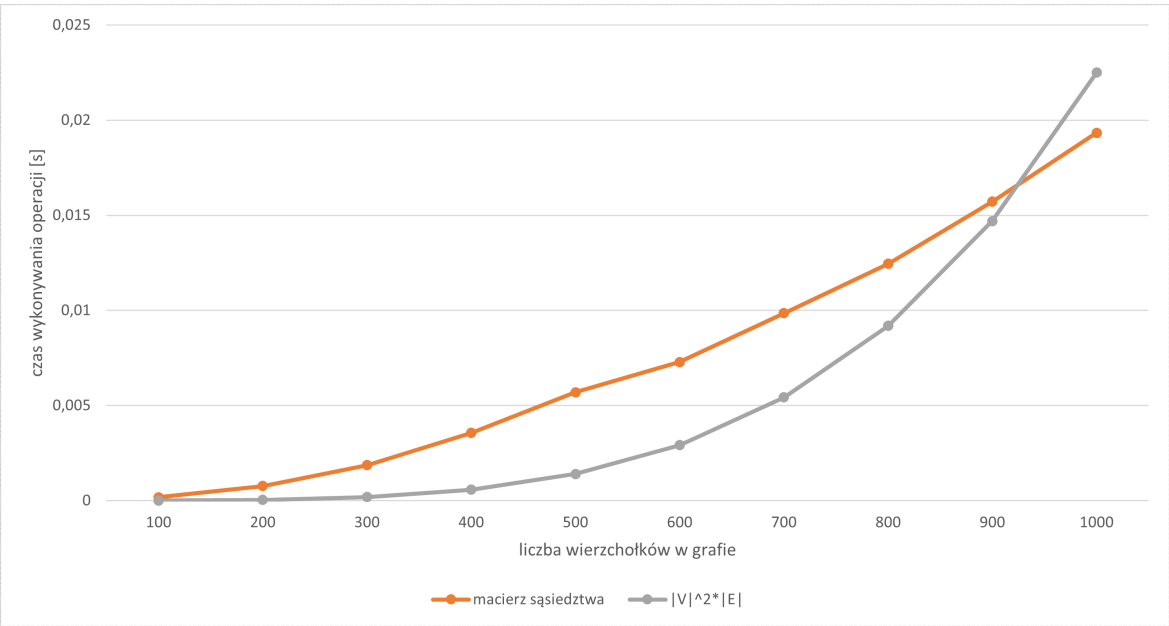
Rysunek 8: Wpływ liczby wierzchołków w grafie o gęstości 0,3 na czas wykonywania algorytmu Jarnika-Prima w grafie na reprezentacji listy sąsiedztwa.



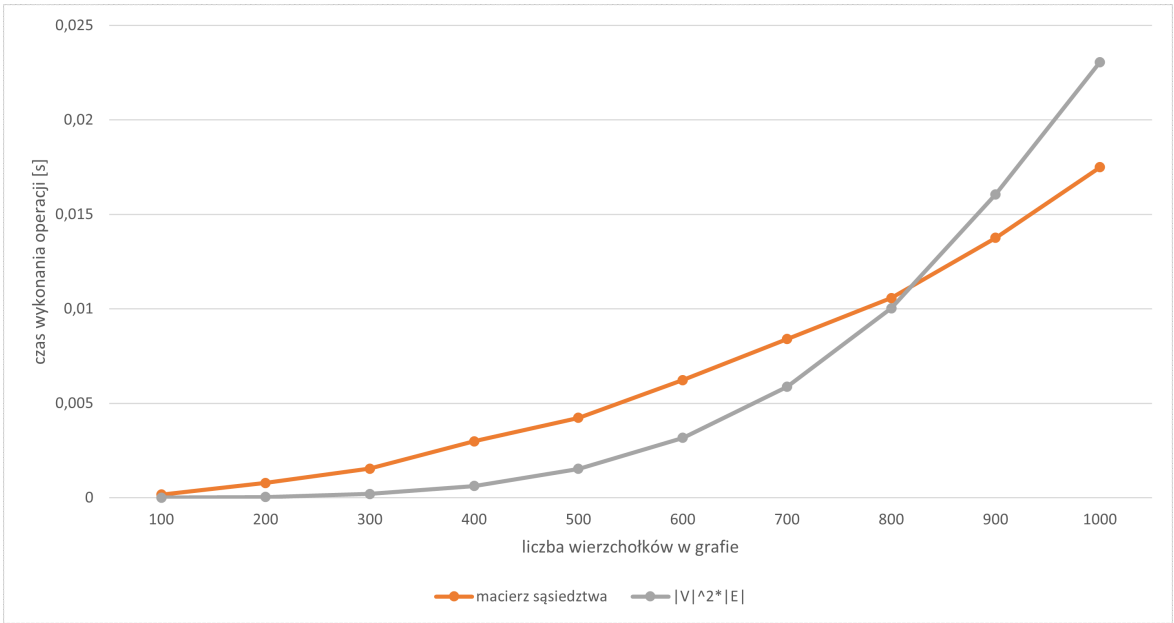
Rysunek 9: Wpływ liczby wierzchołków w grafie o gęstości 0,7 na czas wykonywania algorytmu Jarnika-Prima w grafie na reprezentacji listy sąsiedztwa.

5.2 Algorytm Dijkstry

5.2.1 Macierz sąsiedztwa

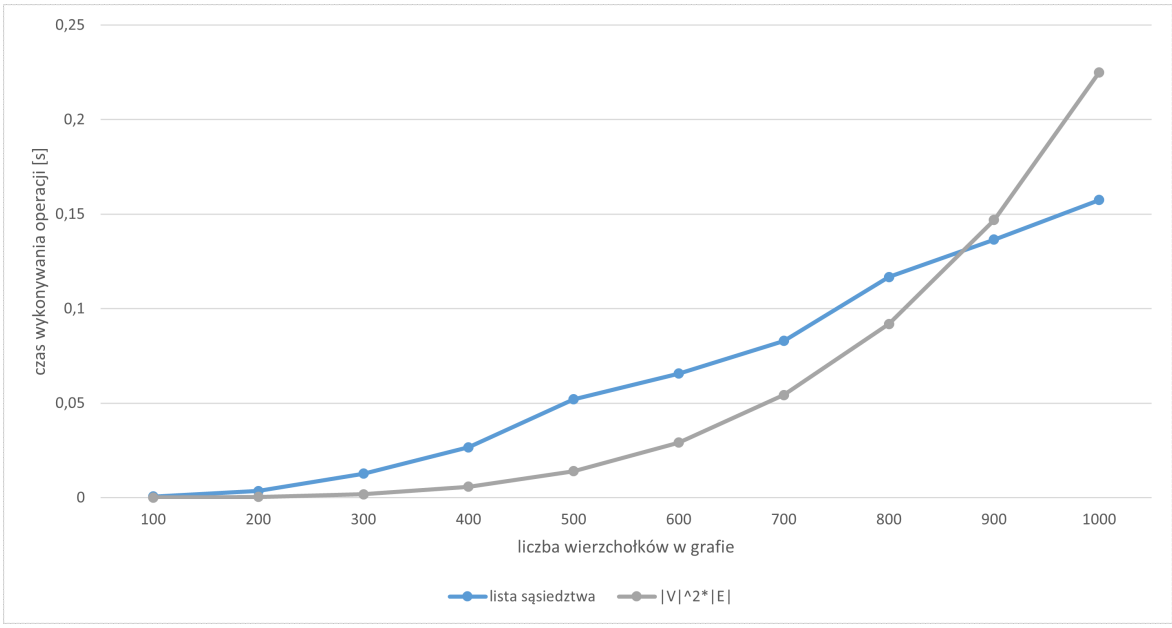


Rysunek 10: Wpływ liczby wierzchołków w grafie o gęstości 0,3 na czas wykonywania algorytmu Dijkstry w grafie na reprezentacji macierzy sąsiedztwa.

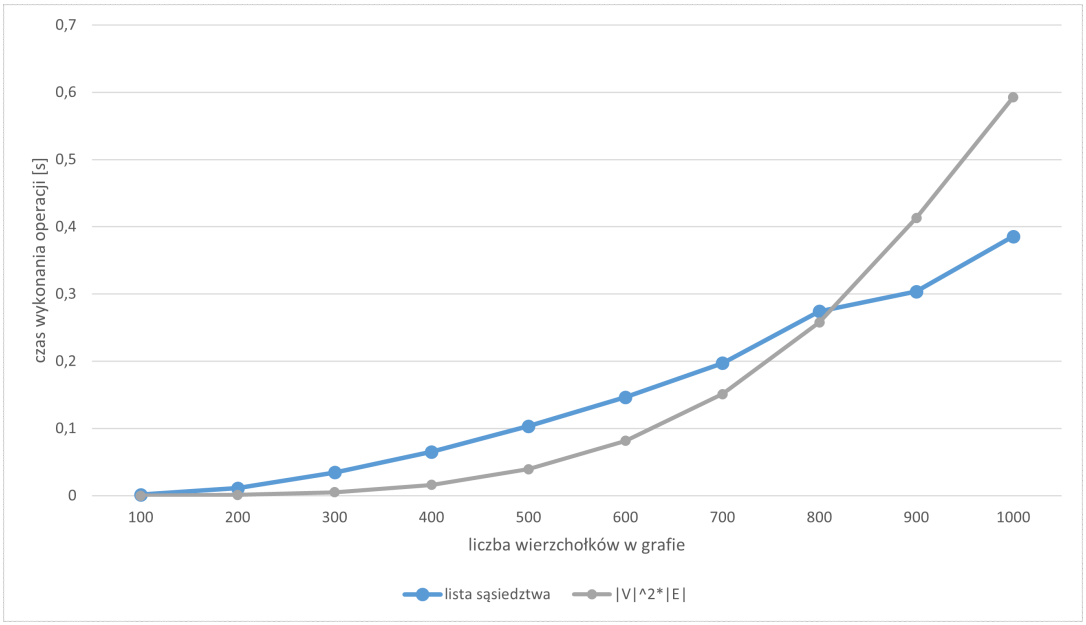


Rysunek 11: Wpływ liczby wierzchołków w grafie o gęstości 0,7 na czas wykonywania algorytmu Dijkstry w grafie na reprezentacji macierzy sąsiedztwa.

5.2.2 Lista sąsiedztwa



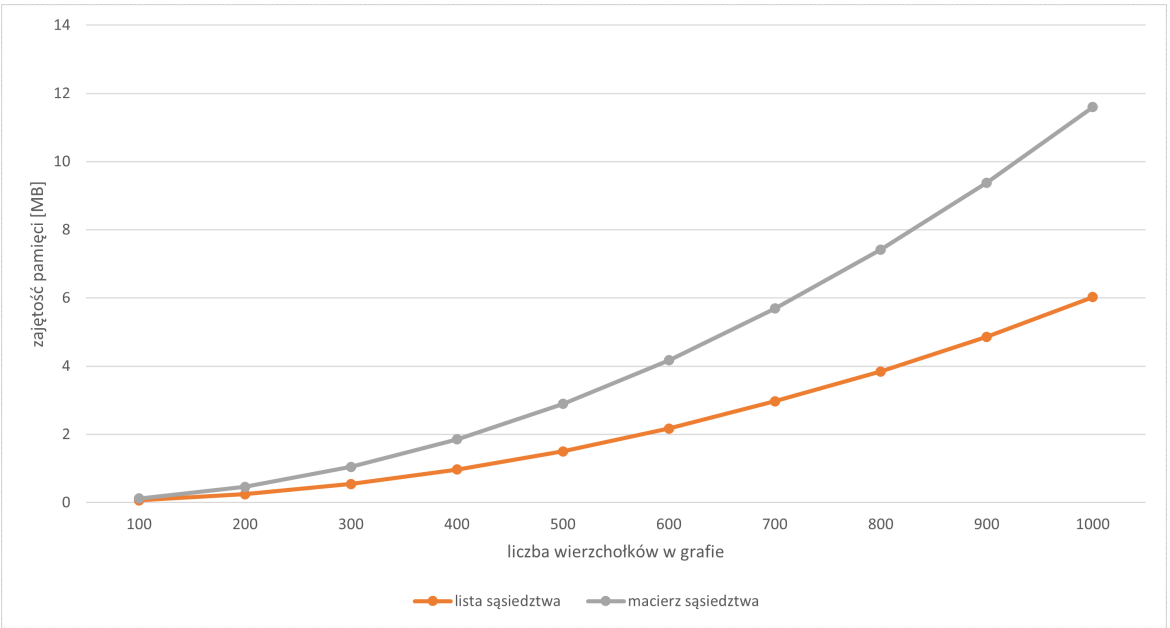
Rysunek 12: Wpływ liczby wierzchołków w grafie o gęstości 0,3 na czas wykonywania algorytmu Dijkstry w grafie na reprezentacji listy sąsiedztwa.



Rysunek 13: Wpływ liczby wierzchołków w grafie o gęstości 0,7 na czas wykonywania algorytmu Dijkstry w grafie na reprezentacji listy sąsiedztwa.

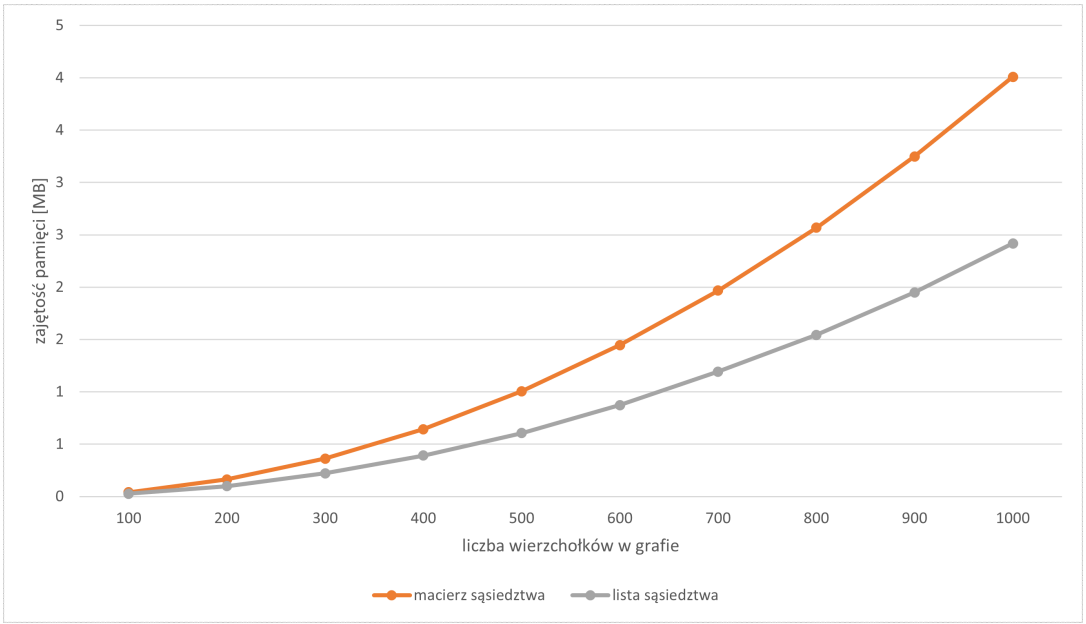
6 Wyniki pomiarów zajętości pamięci podczas wykonywania algorytmów

6.1 Algorytm Jarnika-Prima



Rysunek 14: Wpływ liczby wierzchołków w grafie o gęstości 0,3 na zajętość pamięci podczas wykonywania algorytmu Jarnika-Prima.

6.2 Algorytm Dijkstry



Rysunek 15: Wpływ liczby wierzchołków w grafie o gęstości 0,3 na zajętość pamięci podczas wykonywania algorytmu Dijkstry.

7 Badanie wpływu ujemnych wag na poprawność wykonywania algorytmu Dijkstry

Wykonanie zaimplementowanego algorytmu Dijkstry na badanym grafie daje następujący rezultat:

```
========SHORTEST PATHS=========
INDEX       |  0  1  2  3  4  5
DISTANCE    |  0  5 10  8 11 10
PREDECESSOR |  0  0  4  1  3  3
```

Jak można zauważyć nie we wszystkich wierzchołkach wyznaczona ścieżka jest najkrótszą, co można sprawdzić wykonując algorytm Bellmana-Forda.

Ścieżka do wierzchołka 5 zgodnie z algorytmem ma wagę 10 i następujący przebieg $0 \rightarrow 1 \rightarrow 3 \rightarrow 5$.

Rzeczywista najkrótsza ścieżka do wierzchołka to $0 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 5$ i ma ona wagę 6.

8 Analiza wyników i wnioski

8.1 Algorytm Jarnika-Prima

8.1.1 Macierz sąsiedztwa

Krzywa najlepszego dopasowania zależności wzrostu czasu wykonywania algorytmu Jarnika-Prima pokrywa się z krzywą zależności $|E| * |V|$ (rysunek 6 oraz 7), co potwierdza założenie teoretyczne, że górne ograniczenie złożoność czasowej algorytmu Jarnika-Prima jest równe $O(|E| * |V|)$.

Zależność ta została zbadana na grafach o różnych gęstościach. Można zauważyć, że w grafie o mniejszej gęstości wraz ze wzrostem liczby wierzchołków w grafie czas wykonywania algorytmu zaczyna być mniejszy od górnego ograniczenia złożoności czasowej.

8.1.2 Lista sąsiedztwa

Krzywa najlepszego dopasowania zależności wzrostu czasu wykonywania algorytmu Jarnika-Prima pokrywa się z krzywą zależności $|E| * |V|$ (rysunek 8 oraz 9), co potwierdza założenie teoretyczne, że górne ograniczenie złożoność czasowej algorytmu Jarnika-Prima jest równe $O(|E| * |V|)$.

Zależność ta została zbadana na grafach o różnych gęstościach. Można zauważyć, że w grafie o większej gęstości wraz ze wzrostem liczby wierzchołków w grafie czas wykonywania algorytmu zaczyna być mniejszy od górnego ograniczenia złożoności czasowej.

8.2 Algorytm Dijkstry

8.2.1 Macierz sąsiedztwa

Krzywa najlepszego dopasowania zależności wzrostu czasu wykonywania algorytmu Dijkstry pokrywa się z krzywą zależności $|V|^2 * |E|$ (rysunek 10 oraz 11), co potwierdza założenie teoretyczne, że górne ograniczenie złożoność czasowej algorytmu Jarnika-Prima jest równe $O(|V|^2 * |E|)$.

Zależność ta została zbadana na grafach o różnych gęstościach. Można zauważyć, że w grafie o większej gęstości wraz ze wzrostem liczby wierzchołków w grafie krzywa wzrostu czasu wykonywania algorytmu zaczyna być bardziej zbliżona do górnego ograniczenia złożoności czasowej.

8.2.2 Lista sąsiedztwa

Krzywa najlepszego dopasowania zależności wzrostu czasu wykonywania algorytmu Dijkstry pokrywa się z krzywą zależności $|V|^2 * |E|$ (rysunek 12 oraz 13), co potwierdza założenie teoretyczne, że górne ograniczenie złożoność czasowej algorytmu Jarnika-Prima jest równe $O(|V|^2 * |E|)$.

Zależność ta została zbadana na grafach o różnych gęstościach. Można zauważyć, że w grafie o większej gęstości wraz ze wzrostem liczby wierzchołków w grafie krzywa wzrostu czasu wykonywania algorytmu zaczyna być bardziej zbliżona do górnego ograniczenia złożoności czasowej.

8.3 Złożoność pamięciowa algorytmów

8.3.1 Algorytm Jarnika-Prima

Analizując przebieg algorytmu można zaważyć, że w czasie działania algorytmu Jarnika-Prima konieczne jest dwukrotne stworzenie struktury reprezentującej graf (jedna dla grafu w którym szukane jest minimalne drzewo rozpinające, druga do zapisania minimalnego drzewa rozpinającego), stworzenie tablicy wierzchołków odwiedzonych oraz kolejki priorytetowej.

Złożoności będą się różnić w zależności od zastosowanej struktury i jak zostało to pokazane na wykresie (rysunek 14) zajętość pamięci w przypadku zastosowania macierzy sąsiedztwa jest prawie dwukrotnie większe niż w przypadku listy sąsiedztwa. Wynika to z gęstości grafu - rozmiar listy sąsiedztwa jest determinowany w dominującym stopniu przez liczbę krawędzi, natomiast rozmiar macierzy sąsiedztwa jest determinowany wyłącznie przez liczbę wierzchołków. Wraz ze wzrostem gęstości grafu zajętości pamięci dla obu struktur powinny się zbiegać. Przy bardzo dużych gęstościach macierz sąsiedztwa może okazać się korzystniejszą strukturą ze względu na brak nadmiarowych danych, które znajdują się w liście sąsiedztwa (wskaźniki na kolejne elementy w liście).

Na podstawie powyższej analizy można wyciągnąć wniosek, że górne ograniczenie złożoności pamięciowej algorytmu Jarnika-Prima wynosi $O(|V|^2 * |E|)$

8.3.2 Algorytm Dijkstry

Analizując przebieg algorytmu można zaważyć, że w czasie działania algorytmu Dijkstry konieczne jest utworzenie struktury reprezentującej graf w którym szukamy najkrótszych ścieżek, tablicy wierzchołków odwiedzonych oraz tablicę przechowującą listę wag ścieżek oraz poprzedników wierzchołków.

Podobnie jak w przypadku algorytmu Jarnika-Prima istnieje różnica w zajętości pamięci ze względu na wykorzystaną strukturę reprezentującą graf co zostało opisane w powyższym akapicie.

Na podstawie powyższej analizy można wyciągnąć wniosek, że górne ograniczenie złożoności pamięciowej algorytmu Dijkstry wynosi $O(|V|^2)$

8.4 Wpływ ujemnych wag na poprawność wykonywania algorytmu Dijkstry

Analiza działania algorytmu Dijkstry na przykładzie grafu posiadającego ujemne wagi potwierdziła założenia teoretyczne, że algorytm ten nie ma zastosowania w przypadku grafów z ujemnymi wagami. Wynika to z faktu, że wyznaczona najkrótsza ścieżka to rozpatrzonego wierzchołka jest niezmieniana w dalszym działaniu algorytmu.

W celu wyznaczenia najkrótszych ścieżek w takim grafie należy użyć algorytmu Bellmana-Forda.

8.5 Wnioski

Sumując wyciągnięte wnioski można określić wyróżnić następujące wnioski:

- Reprezentacją odporną na zmiany gęstości grafu jest macierz sąsiedztwa. Zarówno macierz incydencji jak i lista sąsiedztwa swój rozmiar uzależniają od liczby krawędzi, a więc gęstości grafu (przy założeniu stałej liczby wierzchołków). Niekoniecznie oznacza to, że macierz incydencji czy lista sąsiedztwa są gorszymi strukturami - w przypadku dużych grafów o małej gęstości mogą okazać się bardziej optymalnym rozwiązaniem

- Biorąc pod uwagę trudności napotkane w procesie implementacji najtrudniejszą strukturą w implementacji jest macierz incydencji. Największą trudnością było przełożenie koncepcji samej struktury macierzy incydencji na stryktury danych dostępne w języku programowania C++. Najprostszą strukturą okazała się być macierz sąsiedztwa co wynika z intuicyjnych właściwości tej struktury.

- Ciężko jednoznacznie określić która z badanych struktur jest najefektywniejsza czasowo lub pamięciowo. W większości przypadków, gdy zależy twórcy algorytmu na szybkim czasie działania, lista sąsiedztwa może okazać się najszybszą strukturą, ze względu na szybki dostęp do wszystkich krawędzi wychodzących z danego wierzchołka. Również pod względem pamięciowym może ona kazać się najefektywniejsza, ponieważ jej rozmiar jest determinowany przez liczbę krawędzi. Jeżeli mamy na celu optymalizację pamięciową również macierz incydencji może okazać się efektywna i nawet wydajniejsza niż lista sąsiedztwa, jednak nie została ona dokładnie zbadana w tym projekcie.